マイクロコンピュータとワークステーション 64－2
（１９９０．　１１．　３０）

宇宙機搭載用コンピュータのためのフォールトトレラントＯＳ

金川　信康†，井原　廣一‡，加藤　肇彦‡

†㈱日立製作所　日立研究所　　‡㈱日立製作所　宇宙技術推進本部

フォールトトレランス機能を持つオペレーティングシステム(ＯＳ)を開発した．本ＯＳでは冗長系間の処理の時間ずれから生じるデータ処理結果の矛盾の発生を防ぐためにメイルボックスによるバッファリング等のＩＴＲＯＮ仕様ＯＳ，ＨＩ６８Ｋに備わっているタスク間通信同期機能のほかに，新たに開発したランデブー方式などを用いて冗長系間の同期を採る機能を実現した．タスク間のデータの交換方式には汎用性を持たせ，カーネルとしてのＩＴＲＯＮ仕様ＯＳの採用と相俟って種々の用途への応用を可能にし開発成果の再利用性を高めた．

Operating System for Fault-Tolerant Onboard Space Computers

Nobuyasu KANEKAWA†, Hirokazu IHARA‡, and Hatsuhiko KATOH‡

†Hitachi Res. Lab., Hitachi, Ltd.,
4026 Kuji-cho, Hitachi -city, IBARAKI 319-12 Japan
Phone: 0294-52-5111 ext.3821
E-mail:kanekawa@hrl.hitachi.co.jp

‡Space Systems Div., Hitachi, Ltd.

The Authors developed the operating systems(hereinafter OS) with fault-tolerance. In this OS, newly developed skill called rendezvous in addition to the ITRON spec. OS - HI68K's inter-task syncronization and communication function such as data buffering in mailbox for consistency. In addition to usage of ITRON spec. OS as its kernel, standarization of data exchange among tasks accomplished high versatility.

1．まえがき

交通管制や電力システム等の社会の中枢ともいえる機能をコンピュータが担うにつれて，益々信頼性がコンピュータに要求されている．

また，宇宙や海洋などの極限環境下で人間に代わって作業し，あるいは人間の活動を支援する為のコンピュータが登場している。これらのコンピュータには高い信頼性と共に耐環境性が要求されている．[1)-3)]

高い信頼性を持つフォールトトレラントコンピュータを構成するために，同一の機能をもつモジュールを複数用意しておき(冗長化)，それらの中から適切な出力を選択する方法が一般に採られている．この場合には冗長系間のデータのコンシステンシーの維持が重要な課題となる．報告者らは，フォールトトレランスのための処理，コンシステンシーの維持及びこれに必要な同期のための機能を68000用のＩＴＲＯＮ*仕様ＯＳ，ＨＩ６８Ｋをカーネルにして開発したのでここに報告する．

2．フォールトトレランスＯＳの機能

(1) フォールトのマスク

フォールトトレランスＯＳの最大の機能は，フォールト発生の影響がシステム全体の動作に及ばないようにし，システムが誤った信号を出力しないようにすることである．その為には，図2.1に示すように同一の機能を持った冗長系を複数用意しておき，それらの出力の中から適切な出力を選択する方法が採られる．

図2.1のように冗長系間で交換した処理結果を比較する相互診断の結果と，冗長系間で交換した自己診断結果を総合的に用いて各冗長系の出力の信頼度を推定し，最も信頼度の高い出力を選択するＳＮＶ(Stepwise Negotiating Voting)方式を報告者らは提案している．[4),5)]

＊TRONは"The Real Time Operating System Nucleus"の略である．

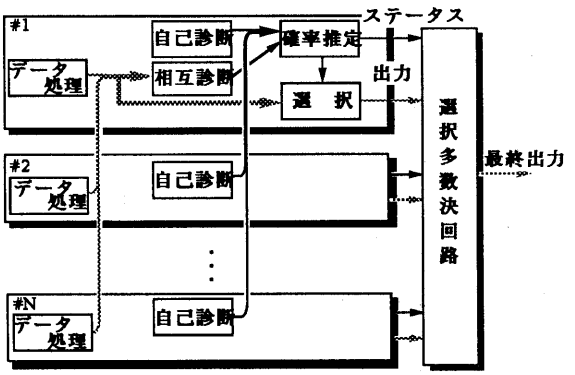

図2.1　ハードウェア構成

(2) 冗長系間の同期

データ処理が正常に進行している場合でも，異なる時刻のデータ同志の一致は保証されない．

例えば，図2.2のように冗長系Ｉではデータ処理が終了したが，冗長系IIではまだ終了していない場合，冗長系Ｉの処理済みのデータと冗長系IIの未処理のデータの間で不一致が生じてしまう．そこで，本システムのような冗長系ごとにクロックが独立している疎結合系ではフォールト・トレランスのために使うデータを冗長系間で交換する際にそれぞれの冗長系の間で，処理の同期を採り，冗長系間でデータのコンシステンシーを維持する必要がある．

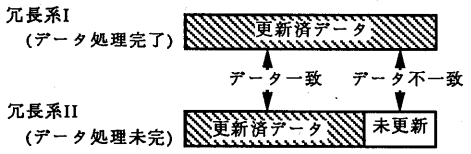

図2.2　タイミングずれによるデータ不一致

また本システムは自律分散の概念を取り入れ，システム全体の動作を司るマネージャの

役割をする部分がないシステム構成を採っている．そのため，競合及びデッドロック(3すくみ)現象の対策が必要である．

(3) フォールトからの回復(システム再構成)

フォールトの発生により一部または全部の冗長系におけるデータ処理が影響を受けた場合，速やかに冗長系の動作を停止し（セイフティ・シャットダウン）し，速やかにダウン状態から回復し正常動作に戻す機能である．他の冗長系が正常に動作している場合にはその冗長系より必要なデータを受け取り，そのデータを用いて自分自身を立ち上げ，システムを再構成し正常動作に移る．

(4) 自己の保護

他の冗長系の異常動作により自系が障害を受けないように保護をする．他の冗長系の異常動作の特に深刻な影響は，頻繁な通信要求により割込み処理に処理時間を取られ，他の処理をする時間が無くなり自分自身がシステムダウンに致ることである．このような影響を防ぐため，他の冗長系の動作を監視し，異常な動作(頻繁な通信要求など)を検出した場合，一定時間当該冗長系との接続をハードウェア的に切断する．

3．フォールトトレランスＯＳの開発

3．1 開発方針

フォールトトレランスＯＳの開発方針を以下に挙げる．

(1) 既存のＯＳの活用

フォールトトレランスＯＳの標準化と開発工程の削減のためにカーネル部分には既存のＯＳを活用することにした．新たにカーネルを開発するより，これらの再利用頻度の高いリアルタイムＯＳをカーネルに使用した方が製造上の誤り(バグ)を容易に排除でき，信頼性を向上できよう．

本開発では，わが国独自の設計思想，入手の容易性，わが国産業界への影響を考慮してＩＴＲＯＮ仕様リアルタイムＯＳ，ＨＩ６８Ｋを使用した．

(2) トランスペアレンシー

システムにアプリケーションタスクからのトランスペアレンシーを持たせ，アプリケーションタスクのフォールトトレラント機能の実現への関与を最小限にする．このため，フォールトトレラント機能は全てＯＳに組み込んでしまい，アプリケーション・タスクでは全くフォールトトレラントのことを考慮しなくても済むようにする．冗長系間の同期への関与は，アプリケーション・タスク実行のために必要なチェックポイントの設定のみとした．

(3) 汎用性

標準性と再利用可能性向上のために，種々のアプリケーションタスクに対応して使用できるようにした．アプリケーション・タスクの種類に依存せずに使用できるようにするため，アプリケーション・タスクとフォールトトレランスＯＳとの間のデータの受授などのインタフェースについては特に汎用性を考慮して開発した．

ソフトウェアが再利用可能であるということは，単にソフトウェアの生産性が高いということに留まらず，ソフトウェアの信頼性の向上につながる．再利用可能なソフトウェアは繰返し使用され，その度に使用実績が蓄積されてゆく．もしもソフトウェアをハードウェアと同様に十分に試験された高信頼性部品(ソフトウェアパッケージ)を組み合わせて作ることができればソフトウェアの信頼性は飛躍的に向上しよう．

3．2 構成

(1) システム構成

ソフトウェアは，図3.1に示すようにカーネル，準カーネル，アプリケーションから構成される．

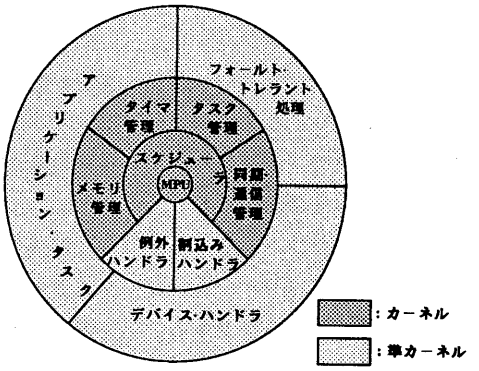

図3.1　ソフトウェアの構成

カーネルはオペレーティング・システムの中核をなすもので，本開発では開発費の削減，開発期間の短縮のためにカーネルは既存のオペレーティング・システムのものを使用した．準カーネルは例外ハンドラ，割込みハンドラ，ディバイス・ハンドラ，フォールト・トレラント・タスクから構成される．これらの準カーネルの構成モジュールは新たに開発した部分である．

(a) 例外ハンドラ，割込みハンドラ

例外ハンドラはバスエラー，アドレスエラーなどのＭＰＵの例外割込みに対する処理や，システムコールのエラー処理を実行する．

割込みハンドラは，外部割込み発生時に起動されるハンドラで，割込み処理を実行したり，他のタスクに割込み発生を通知する．

(b) ディバイス・ハンドラ

外部との入出力を管理するハンドラで，例えばバブルメモリーハンドラや，サブシステム間通信ハンドラなどがある．ディバイス・ハンドラは外部から入出力要求割込みを受けた割込みハンドラや，外部との入出力を要求する他のタスクによって起動される．

(c) フォールト・トレラント・ハンドラ

各種のフォールト・トレラント処理を実行するハンドラである．本ハンドラを構成するタスクは起動方法によって次のように分類される．

○フォールト・トレラント被起動タスク

アプリケーションタスクにより起動されるタスクである．起動された時点をチェック・ポイントにして，サブシステム間で各種データの照合・更新などをする．ＳＮＶ方式は本タスクで実行される．

○フォールトトレラント周期起動タスク

周期的に起動するタスクである．タスク実行中には他のタスクは動作を停止する．本タスクには以下のように周期的に実行する必要のある処理，他タスクの動作の影響を受けると正常動作しない処理が含まれている．

3．3 同期

図3.2に主要なタスクの動作を示す．

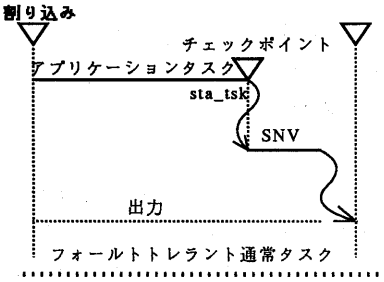

図3.2　主要なタスクの動作

フォールト・トレラント機能の存在を意識しなくともアプリケーション・タスクを作成できるように，アプリケーション・タスクの冗長系間の同期への関与は，チェックポイントでのフォールト・トレラント被起動タスクの起動のみとした．その他の冗長系間の同期はフォールトトレランスＯＳ内部でメイルボックスによるバッファリングやランデブー方式を用いてフォールトトレランスＯＳ内部で採るようにした．表3.1にコンシステンシの維持，同期の方法をまとめる．

表3.1　コンシステンシの維持，同期の方法

用途＼同期の方法		チェックポイント	メイルボックス	ランデブー	他タスク動作抑制	外部割込
アプリケーションタスク		○	↓			
FTOS	ソフトウェアボーティング	↑	↓			
	システム再構成	○	↓			
	履歴データ照合訂正		↓			
	フィードバックデータチェック		↓	○		
	冗長系間通信		○			
	多重化データ照合				○	
	入　力					○
	出　力					○

(1) メイルボックスによる同期

例えば冗長系間での処理の進行状況に差が生じ，図3.3のように冗長系Ｉが冗長系IIより早く処理Ａを終了した場合には以下の様にしてメイルボックスにより同期を採る．

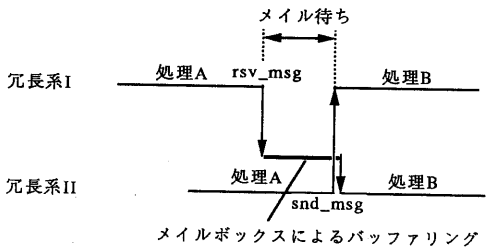

図3.3　メイルボックスによる同期

冗長系Ｉでは処理Ａを終了した後，冗長系IIから処理Ａの結果のデータがメイルとして送られてくるのを待ち，データを受け取った後に処理Ｂを開始する．このメイル待ちにはタイムアウトを設け，冗長系IIがダウンしてメイルが送られてこなかった場合でも，タイムアウト設定時間後には処理Ｂを開始して，冗長系Ｉのダウンを防止している．　一方，冗長系IIでは冗長系Ｉからデータを処理Ａ終了前に受け取っているため，処理Ａが終了するまでメイルボックスによりデータをバッファリングする．

ここでは，rcv_msg，snd_msgシステムコールを使用している．

(2) ランデブー

冗長系間で，同時に同一の処理を開始するためにランデブー方式により同期を採る．

図3.4は，同期が不要な処理Ａのあと，ランデブー方式により同期を採り，同期が必要な処理Ｂを開始する様子を示したものである．

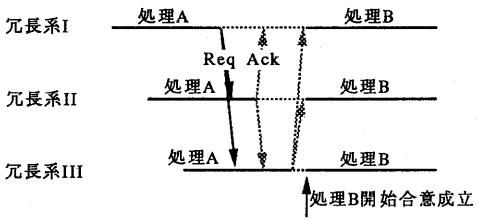

図3.4　ランデブー方式による同期

先に処理Ａを終えた冗長系Ｉはランデブーのためのマスタになり，冗長系II，冗長系IIIに処理Ｂ開始のリクエスト信号(Req)を送る．冗長系II，冗長系IIIでは，処理Ａを終えランデブー処理に入ると，冗長系Ｉからのリクエストに対してアクナレッジ信号(Ack)を返送する．所定の数のアクナレッジを受け取った時点で処理Ｂ開始のための同意が成立し，処理Ｂが開始される．

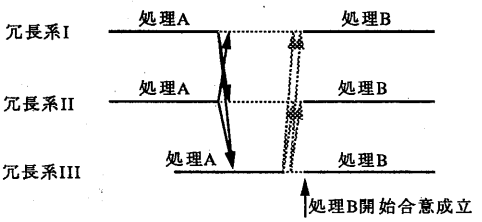

図3.5　ランデブー方式による同期
（２つの冗長系がマスタ）

また，先に処理Ａを終えた冗長系が２つ以上ある場合にはそれらの冗長系がマスタになる．図3.5に２つの冗長系がマスタになった場合のランデブーの様子を示す．

なお，自分が先に処理Ａを終えたかどうか（マスタかどうか）は，ランデブー処理の開始時に既に他の冗長系よりリクエストが到着しているかどうかで判定できる．

合意成立に必要なアクナレッジの数とリクエストの数との間には次のような関係がある．

(a) 自分がマスタであるとき

$$Nack = (N - Nreq - 1)\ (Nreq + 1) \quad \cdots\cdots(3\text{-}1)$$

ただし，Nack：アクナレッジの数

Nreq：リクエストの数

N：全サブシステム数

(b) 自分がマスタでないとき

$$Nack = (N - Nreq - 1)\ Nreq \quad (3\text{-}2)$$

なお，これらの式の導出については，付録で述べる．

一部の冗長系が障害などのためにアクナレッジを送れない場合には，他の冗長系はタイムアウトにより処理Ｂを開始する．この場合，図3.6のように全ての正常な冗長系からアクナレッジが到着した後指定したタイムアウト時間後に処理Ｂ開始の合意が成立する．

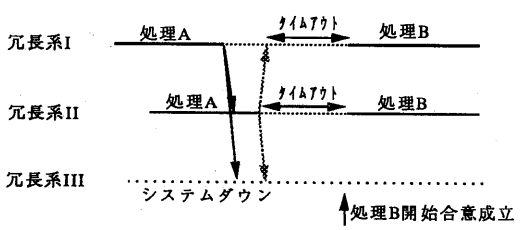

図3.6　ランデブー方式による同期
（冗長系III障害）

3．4 データ交換

(1) 冗長系間通信

自律分散の原理に従い，冗長系間に絶対的な優先順位が生じるのを避けるため，個々の冗長系は同一のアルゴリズムに従って動作する．もし，冗長系間に絶対的な優先順位が存在すると，優先順位の高い冗長系の信頼性が系全体の信頼性を左右してしまうからである．従って，冗長系間の通信でも優先順位は絶対的にではなく，常に相対的に決定される．例えばループ型のネットワークで個々の冗長系が中心に向かって右側の冗長系への通信を優先させるアルゴリズムに基づいて動作すると，冗長系間の同期が採れている場合には図3.7のように全ての冗長系が反時計廻りに通信要求を出すことになる．

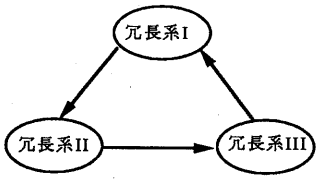

図3.7　通信要求

このとき通信方式が半２重（送信と受信を交互にする）方式であると，全ての冗長系が通信要求を送信しているために，どの冗長系も送信された通信要求を受信することができないためにデッドロックに陥ってしまう．デッドロック発生時の再試行（リトライ）の時間間隔も絶対的な優先順位付けを避けるために全ての冗長系とも同一にしなければならないため，デッドロックの発生を繰り返してしまう．このようなデッドロック発生を防ぐために，冗長系間の通信は全２重（送信と受信が同時にできる）方式とした．

(2) データの授受

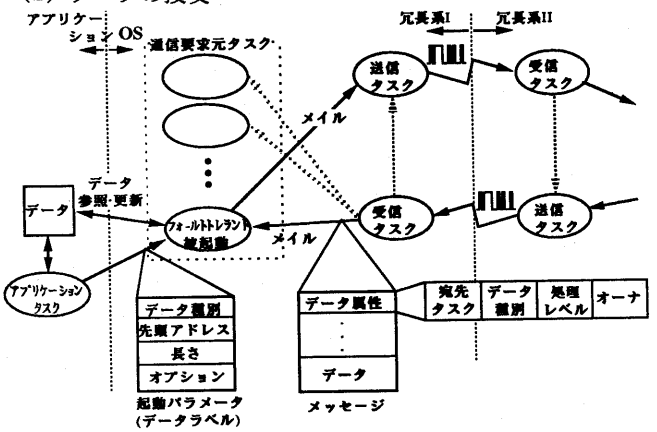

図3.9 データ交換

図3.9にデータ交換の方法を示す．アプリケーション・タスクからフォールト・トレラント披起動タスクへのデータの受け渡しは，図のようにデータ・ラベルを起動パラメータとして渡す．データ・ラベルは多数決を採るデータのデータ種別，先頭アドレス（ポインタ），データ長などを表している．フォールト・トレラント披起動タスクでは，データ・ラベルをもとに他の冗長系とデータを交換し，自身のデータが誤っている場合には，データ・ラベルにより示された領域に他の冗長系から受け取った正しいデータを書き込む．このようにデータをラベルで渡すことにより任意の形式のデータを扱うことができる．

送信，受信タスクとの間では，メイルボックスを用いてデータをメイルとして交換する．メイルには交換すべきデータの他に，データの属性として宛先タスク名，データ種別，処理レベル，オーナ等の情報が含まれている．

4．まとめ

ＩＴＲＯＮ仕様ＯＳ，ＨＩ６８Ｋをカーネルにしてフォールトトレランスを実現するためのＯＳを開発した。

アプリケーションタスクのフォールト・トレラント機能への関与の低減，タスク間で交換するデータ形式の標準化に留意して開発したために，フォールトトレランスＯＳの汎用性を高めることができた．

図4.1の様にチェックポイントでフォールト・トレラント被起動タスクを起動し，その際に起動パラメータとしてデータラベルをフォールト・トレラント被起動タスクにわたせば，各種のアプリケーションにおいて容易にフォールトトレラント機能を実現できる．

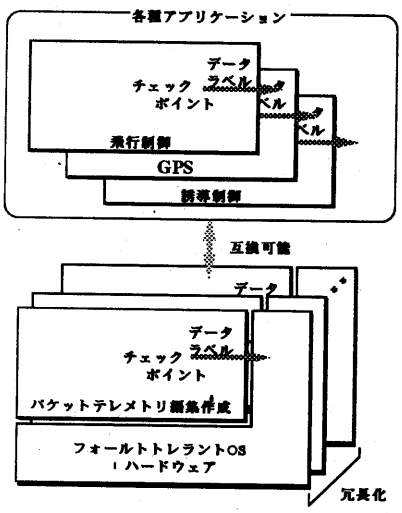

図4.1 フォールトトレランスＯＳの汎用性

参考文献

1) 山田　弘善ほか：電子機器に対する宇宙環境，信学誌，66，3，pp.254-258 (1983)

2) 高野　忠ほか：耐故障性を有する人工衛星搭載用コンピュータの開発，信学技報，SANE86-43，pp.25-30 (1986)

3) T. Takano et al.,:Fault-Tolerant Onboard Computers, Proc. 16th Int'l Symp. Space Tech. and Sci., ISTS-16, Sapporo, pp.1097-1100 (1988-06)

4) 金川　信康ほか：”汎用性を持たせた多数決の方法に関する一提案”,信学技報、FTS88-36, pp.65-70 (1989-02)
5) N. Kanekawa et al., "Dependable Onboard Computer Systems with a New Method - Stepwise Negotiating Voting," Proc. 19th Fault-Tolerant Computing Symp., FTCS-19, pp.13-19 (1989)
6) 金川　信康ほか：新しい多数決方式によるフォールトトレラントコンピュータシステム，信学論，J73-D-I, 2, pp.109-116 (1990)
7) Tadashi Takano et al. "Longlife Dependable Computers for Spacecrafts," IFIP Int'l Working Conf. Dependable Computing for Critical Applications, (1989)
8) 加藤　肇彦，石川　知雄：マイクロコンピュータ用オペレーティングシステムの動向，情報　処理，20, 6, pp.717-724 (平1-6)
9) ＨＩ６８Ｋユーザーズマニュアル，㈱日立製作所 (昭62-3)
10)加藤　肇彦，茶木　英明：新マイクロプロセッサH32とその開発環境，情処研究報告，89-MC-56-3, (1989)

付　録

アクナレッジの数とレクエストの数の関係

（ａ）自分がマスタのとき

自分自身は，自分以外のマスタが出したリクエストを受け取るから，受け取るリクエストの数Ｎreqは，

$$Nreq = Nm - 1 \qquad (A\text{-}1)$$

ただし，Ｎｍ：マスタの数

となる．

それぞれのスレーブ（マスタにならなかった冗長系）は，マスタから受け取ったリクエストの数だけアクナレッジを出すから，受け取るアクナレッジの数は，

$$Nack = Ns \cdot Nm \qquad (A\text{-}2)$$

ただし，Ｎｓ：スレーブの数

となる．マスタ以外の冗長系はスレーブであるから，

$$Ns = N - Nm \qquad (A\text{-}3)$$

となる．

ここで，式(A-1)，(A-2)，(A-3)から，

$$Nack = (N - Nreq - 1)(Nreq + 1) \qquad (A\text{-}4)$$

となる．

（ｂ）自分がスレーブのとき

自分自身は全てのマスタの出したリクエストを受け取るから，

$$Nreq = Nm \qquad (A\text{-}5)$$

となる．

自分自身は，自分以外のスレーブの出したアクナレッジを受け取り，それぞれのスレーブ（マスタにならなかった冗長系）は，マスタから受け取ったリクエストの数だけアクナレッジを出すから，受け取るアクナレッジの数は，

$$Nack = (Ns - 1) \cdot Nm \qquad (A\text{-}6)$$

となる．ここで，式(A-5)，(A-6)，(A-3)から，

$$Nack = (N - Nreq - 1)Nreq \qquad (A\text{-}7)$$

となる．